Sesión 19: Genéricos

Programación 2

Ángel Herranz

2020-2021

Universidad Politécnica de Madrid

En capítulos anteriores

- Tema 1: Intro a POO
- Tema 2: Clases y Objetos y TADs y módulos
- Tema 3: Colecciones con *arrays*
- Tema 4: Cadenas simplemente enlazadas
- Tema 5: TAD Listas
- Tema 7: Herencia[1] y Polimorfismo

[1] Sólo de interfaz.

En el capítulo de hoy

- Tema 5: TAD Listas
- Tema 7: Polimorfismo
  - ***Ad hoc***: mencionaremos sobrecarga
  - **Genéricos**: polimorfismo paramétrico
  - **Subtipado**: *todavía no*

Polimorfismo

- "Cuando un mismo elemento puede tomar diferentes formas"
- Por ejemplo: `"Hola"+ 5`
- Por ejemplo: `System.out.format`
- Por ejemplo: `toString()` o `equals(Object o)`
- Por ejemplo: el código de `LinkedListStr` valdría para `LinkedListInt`
- Muy relacionado con los tipos y con los sistemas de tipos

Polimorfismo: varios tipos

On Understanding Types, Data Abstraction, and Polymorphism

Luca Cardelli and Peter Wegner, 1985

Polimorfismo: varios tipos

On Understanding Types, Data Abstraction, and Polymorphism

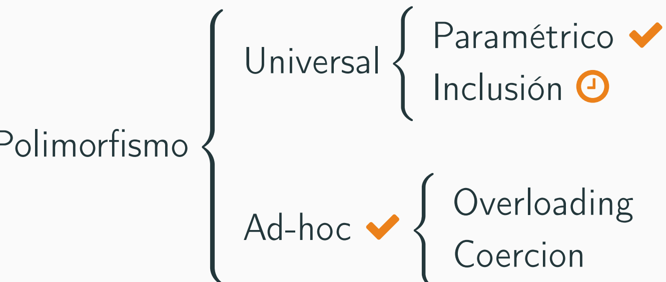
Luca Cardelli and Peter Wegner, 1985

- Polimorfismo
  - Universal
    - Paramétrico
    - Inclusión
  - Ad-hoc
    - Overloading
    - Coercion

Polimorfismo: varios tipos

On Understanding Types, Data Abstraction, and Polymorphism

Luca Cardelli and Peter Wegner, 1985



Ejemplos de poliforfismo Ad-hoc

- Algunos operadores como por ejemplo +:

```
27 + 42 // int x int
"Hola" + " mundo" // String x String
"Lado: " + 5 // String x int
```

- Métodos multi-parámetros:

```
out.format("Hola %s\n", nombre);
out.format("Lado %f, perímetro: %f\n", l, 4*l);
```

- Conversión automática de tipos:

```
27 + 42.0 // 27 se convierte a float
```

Genéricos: polimorfismo paramétrico

Tuplas

- Pero. . . ¿tuplas de qué?

[2] Vale `Integer` $\times$ `Integer`

Tuplas

- Pero. . . ¿tuplas de qué?
- Pongamos que de $\mathbb{Z} \times \mathbb{Z}$[2]

---

[2]Vale `Integer` $\times$ `Integer`

Tuplas

- Pero. . . ¿tuplas de qué?
- Pongamos que de $\mathbb{Z} \times \mathbb{Z}$[2]

```
public class TuplaInt {
  private Integer x;
  private Integer y;

  public TuplaInt(Integer fst,
                  Integer snd) {
    x = fst;
    y = snd;
  }
  public Integer fst() {
    return x;
  }

  public Integer snd() {
    return y;
  }
}
```

[2] Vale `Integer` $\times$ `Integer`

Tuplas de booleanos

```java
public class TuplaBool {
  private Boolean x;
  private Boolean y;

  public TuplaBool(Boolean fst,
                   Boolean snd) {
    x = fst;
    y = snd;
  }

  public Boolean fst() {
    return x;
  }

  public Boolean snd() {
    return y;
  }
}
```

Tuplas de strings

```
public class TuplaString {
  private String x;
  private String y;

  public TuplaString(String fst,
                     String snd) {
    x = fst;
    y = snd;
  }

  public String fst() {
    return x;
  }

  public String snd() {
    return y;
  }
}
```

Tuplas de strings y booleanos

```
public class TuplaStringBool {
  private String x;
  private Boolean y;

  public TuplaStringBool(String fst,
                         Boolean snd) {
    x = fst;
    y = snd;
  }

  public String fst() {
    return x;
  }

  public Boolean snd() {
    return y;
  }
}
```

Tuplas de booleanos y strings

```java
public class TuplaBoolString {
  private Boolean x;
  private String y;

  public TuplaBoolString(Boolean fst,
                         String snd) {
    x = fst;
    y = snd;
  }

  public Boolean fst() {
    return x;
  }

  public String snd() {
    return y;
  }
}
```

Herranz, ¡ya lo he entendido!

Herranz, ¡ya lo he entendido!

```
public class Tupla<T1, T2> {
  private T1 x;
  private T2 y;

  public Tupla(T1 fst,
               T2 snd) {
    x = fst;
    y = snd;
  }

  public T1 fst() {
    return x;
  }

  public T2 snd() {
    return y;
  }
}
```

Vale interpretarlo como una plantilla
en la que substituir **T1** y **T2**

Clase gratis: `Tupla<Integer,Integer>`

```
public class Tupla<Integer, Integer> {
  private Integer x;
  private Integer y;

  public Tupla(Integer fst,
               Integer snd) {
    x = fst;
    y = snd;
  }

  public Integer fst() {
    return x;
  }

  public Integer snd() {
    return y;
  }
}
```

Clase gratis: `Tupla<Boolean,String>`

```java
public class Tupla<Boolean, String> {
  private Boolean x;
  private String y;

  public Tupla(Boolean fst,
               String snd) {
    x = fst;
    y = snd;
  }

  public Boolean fst() {
    return x;
  }

  public String snd() {
    return y;
  }
}
```

Compila y ejecuta (`java -ea ...`)

```
public class PruebaTuplas {
  public static void main(String[] args) {
    Tupla<Integer,Integer> t1 =
      new Tupla<Integer, Integer>(5,1);
    Tupla<Boolean,Boolean> t2 =
      new Tupla<Integer, Integer>(true, false);
    Tupla<String,String> t3 =
      new Tupla<String, String>("Ángel","Herranz");
    Tupla<String,Boolean> t4 =
      new Tupla<String, String>("Ángel",true);
    assert t1.snd().equals(1);
    assert t2.fst();
    assert t3.snd().equals("Herranz");
    assert !t4.snd();
  }
}
```

Entonces. . . ¡Podemos hacer esto!

```
public class Node<T> {
  public T element;
  public Node<T> next;
}
```

¡Y esto!

```java
public interface IList<T> {
  void add(int index, T elem);
  T get(int index);
  int size();
  void set(int index, T elem);
  int indexOf(T elem);
  void remove(int index);
  void remove(T elem);
  IList<T> subList(int start, int end);
}
```

¡Y esto!

```
public class LinkedList<T> implements IList<T> {
  private Node<T> cadena;

  public LinkedList() {
    cadena = null;
  }

  public void add(int index, T elem) {
    ...
  }

  ...
}
```

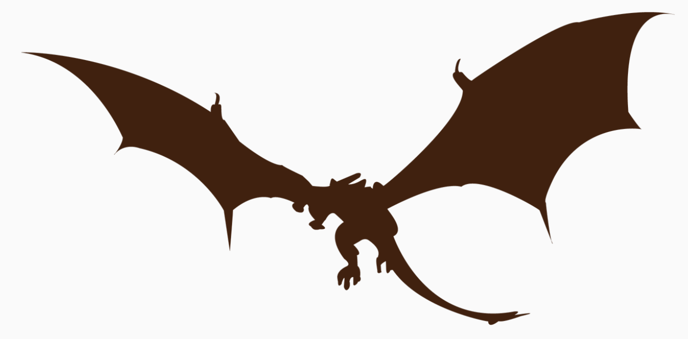
¡Y esto!

```
public class ListStrTest {
  public static void main(String[] args) {
    IList<String> ls = new LinkedList<String>();
    assert ls.size() == 0;
    for (int i = 0; i < N; i++) {
      ls.add(l.size(), "D" + i);
    }
    ...

    IList<Naipe> ln = new LinkedList<Naipe>();
  }
}
```

¡Y esto!

```
public class ListStrTest {
  public static void main(String[] args) {
    IList<String> ls = new LinkedList<String>();
    assert ls.size() == 0;
    for (int i = 0; i < N; i++) {
      ls.add(l.size(), "D" + i);
    }
    ...

    IList<Naipe> ln = new LinkedList<Naipe>();
  }
}
```

There be dragons



¡Cuidado con los genéricos!

Parecen inofensivos pero no lo son.

En especial en presencia de la herencia.